Computer Graphics - 236216

Winter 2024/2025

HW2 – Wireframe renderer & Transformations

Release Date (Week 3): 26/11/2024

Submission Date: 17/12/2024

In this exercise you are required to implement an interactive viewing program for viewing geometric *.itd data files in IRIT format, as wireframe drawings. To allow you to focus your efforts on the graphical part of the project, you will be given the following tools:

1 - A parser for loading the geometric data files, converting the data to polygons, traversing all the geometric data and allowing you to easily implement your code and copy the data into your own data structures. This parser and skeleton will serve you in all forthcoming exercises, so plan ahead as much as you can.

2 - An MFC GUI skeleton.

You will need to implement the matrix transformations (rotation, scaling and translation) discussed in class. Transformations should map each object from object space to a view space. Transformations will be accumulated into a single transformation matrix. The display routine maps the view space to the screen space, which is the current window. You will also support the orthographic and/or perspective projections.

The wireframe edges of the polygons between the mapped vertices will be drawn using the line drawing algorithm discussed in class (either the floating-point algorithm or the Bresenham algorithm as you prefer).

Data Parser

The IRIT parser is already included in the skeleton and its interface resides in the iritSkel.cpp file. You should use the parser to fetch the object's geometry (vertices, triangles, etc.) and attributes (colors, rgb, etc.) and possibly convert it into your own data structure. The parser is called once you open an .itd file. For each object in the 3D model the callback function CGSkelStoreData() will be called by the IRIT parser. Place your code in the marked places in that function.

IRIT data files may contain several types of color information. However in this HW you should use OBJECT color attribute which can be extracted using CGSkelGetObjectColor() and apply this color to all the (line) drawings of this object. Consider that most IRIT files contain multiple objects.

Note that since there is no way for this function to return the newly created data structure to the caller you will have to define your data structure as global variables inside iritSkel.cpp. More information on IRIT data format and the parser can be found on the course website.

GUI

All the source files and a Visual C++ 2019 project solution file are available on the course web site. This project already contains the necessary setup for compiling the source files, adding the IRIT library and PNG image (for the next exercises) support.

The initial GUI in the skeleton program allows the user to perform the following actions, which you will need to implement:

- **Load** - Loads an IRIT geometry data file, *.itd.
- **Set View** - orthographic/perspective. Since this work will also serve you in the forthcoming exercises, we recommend that you employ the perspective warp matrix.
- **Set Transformation** - object/view space (recall this is done by choosing the order of multiplication of the transformation matrices).
- **Transformations**

  o Rotate/Translate/Scale. (pay special attention to scaling in perspective mode)

  o Set active axis (X, Y, Z or mixed XY). For scale also allow **uniform** XYZ scale.

- **Quit** - this option is already implemented.

The skeleton also contains a Menu bar. All its options are already linked to the code, so you only need to implement them. It does the same actions as the buttons in the toolbar, except for the File sub menu. Of course, you can add new menu items/toolbar buttons as needed.

When loading a new IRIT geometry file, your program should initialize the view so that the new object is nicely scaled and positioned in the middle of the screen. Hint: compute a bounding box (see below) of the input geometry.

The skeleton also contains two dialogs which will be used in the next exercise (light and material), so do not change or remove any of them. You can use the material dialog as a reference implementation for your own dialog.

Examples & Documentation

- An example project is available on the course's website. Please note that this example is to help you get a general idea, and not something you should follow. In the case of a discrepancy between the example project and this description, this description will govern.
- In the material section of the web site, you will find many IRIT geometry data files that you can use, along with a viewer that you can compare your work with (and test the integrity of the IRIT files).
- In addition, you will be able to find an IRIT converter – which means that should you wish, you will be able to download from the internet various geometric data files and convert them to IRIT and use them. Supported geometric file formats are 3ds, dxf, igs, obj, off and stl. The converters are not complete, so use them at your own risk!

Basic features

The skeleton includes many functions (most of them are dummy ones). These functions (callbacks) are invoked by the GUI whenever an interaction request is triggered by the user. You are required to implement the following features:

- You should draw lines using the line drawing algorithm discussed in class (either the floating-point algorithm or the Bresenham algorithm as you prefer). Just to be clear, you are not allowed to use the device context methods for drawing lines (such as LineTo).
- Provide a way to control the sensitivity of transformations with respect to mouse movement. High sensitivity means that a small mouse movement will transform the object significantly. Make sure the sensitivity is initialized for each model in an intuitive way (hint: when translating a model, how much it should move for each pixel of a mouse movement in screen space?)
- Add a button to choose in which space you want to transform the object(s) (object/view space).

- If the window is resized (enlarged or reduced), the object should be resized *while maintaining its proportions* (aspect ratio).
- Add a way to see the normals of the polygons of your object. Normals can be provided by the polygons in the IRIT model (in the Plane field, testing if IP_HAS_PLANE_POLY(Poly) != 0 on an IPPolygonStruct Poly) or calculated. The face normal can be calculated for example via a cross product of adjacent, non co-linear, edges in the polygon.
- Add a way to see the normals of the vertices of your object. Normals can be provided by the vertices in the model (in the Normal field, testing if IP_HAS_NORMAL_VRTX(Vrtx) != 0 on an IPVertexStruct Vrtx). If no normal is found, only issue an error message, once, per this model.
- For both a polygon and a vertex, add an option (one menu item for both) to determine whether the calculated normal or those provided by the model (if any, and if not – use the calculated normal, if you do – see also below) are displayed.
- Draw the bounding box of the objects, for each object, in the same color as the object itself, as wireframe as well. The bounding box of an object is the box generated between the coordinates $Q_{min}(x_{min}, y_{min}, z_{min})$ and $Q_{max}(x_{max}, y_{max}, z_{max})$ where Q_{min} is the minimal value of all the coordinates in the object, and Q_{max} is the maximal value. Note: when you translate, rotate, and scale your object, the bounding box is also translated, rotated, and scaled.
- Add a way to control the perspective matrix, i.e. the perspective projection ratio, by modifying the 'd' variable in the matrix (and other variables to your consideration).
- Many objects in Irit are represented by smooth freeform polynomial functions (splines). When loading these objects they are converted into polygons, the number of polygons is controlled by some tessellation tolerance attribute, which is stored in CGSkelFFCState global variable (FineNess member) in iritSkel.cpp file. Provide a way to control this polygon fineness tolerance, the minimal value is 2, and the default is 20.
- Add a way that enables the user to change the wireframe color, the normal color, and the background color of the window (that will override the model's original color).
  Use CColorDialog to pick the colors.

Advanced features

Implementation of the above functionality will yield 80% of your grade.

For the last 20% you are required to add to your program *one of the following* advanced features:

- Add a way to apply transformations to a single object in the scene. That means that a selection mechanism for selecting different objects (== IPObjectStruct) would be needed.
- Add a feature for highlighting (for example, by changing color) the faces below the mouse cursor on mouse click, note that this should work properly in both projection modes.
- Calculate vertex normal for a model that has none. The calculated vertex normal will be defined as an average of the normals of its incident faces (Faces that pass through the vertex). For this you need to maintain connectivity information for each vertex, which should include a list of incident polygons for each vertex. Hint: You can implement it by hashing the coordinates of the vertex up to a fixed accuracy.

Submission:

You are encouraged to start working on it right away, as it is not a trivial exercise. Submit a zip file containing your code at the web site. Do not submit the Release or Debug directories or the CGWork.ncb file. If your zip file takes more than 5 megabytes, you are doing something wrong. In the week following the deadline, we will ask you to meet us to orally present your work. Please make sure you stay updated with the announcements on the course website.

Final notes:

- DO NOT USE any external code **without permission** and neither should you use other IRIT functions without permission. If you have any doubt, please contact the TA.
    - DO NOT USE any external code – i.e. not even a linear algebra library, but rather implement all the appropriate classes for vectors and matrices and their operations.
- The skeleton includes a sample code that uses the MoveTo and LineTo methods of CDC. You are *NOT* allowed to use them, but rather need to implement a line drawing algorithm by yourself, as mentioned above.
- Please check the FAQ section on the course website before you send your questions.
- Frontal submission will be held in the CG lab (at most in pairs). Make sure your code compiles and runs from different directories.
- Late submission should be coordinated with the TA. A penalty of 3 points will apply for each working (Sunday to Friday) day of delay, if not justified, for up to seven working days.

- Submit electronically a single zip file, named <ID1>_{<ID2>_}HW1.zip, where ID1 {and ID2} are your id numbers. The file should contain the following:
    1. A readme.txt file which includes your names, ids and emails.
    2. The whole homework project with the solution files VS201~~7~~9, don't include the intermediate and compilation files in the Release and Debug folders.

Good luck and enjoy!